Turbocharging Heuristics for Weak Coloring Numbers

Alexander Dobler
TU Wien, Austria

Manuel Sorge
TU Wien, Austria

Anaïs Villedieu
TU Wien, Austria

Abstract

Bounded expansion and nowhere-dense classes of graphs capture the theoretical tractability for several important algorithmic problems. These classes of graphs can be characterized by the so-called weak coloring numbers of graphs, which generalize the well-known graph invariant degeneracy (also called k-core number). Being NP-hard, weak-coloring numbers were previously computed on real-world graphs mainly via incremental heuristics. We study whether it is feasible to augment such heuristics with exponential-time subprocedures that kick in when a desired upper bound on the weak coloring number is breached. We provide hardness and tractability results on the corresponding computational subproblems. We implemented several of the resulting algorithms and show them to be competitive with previous approaches on a previously studied set of benchmark instances containing 86 graphs with up to 183831 edges. We obtain improved weak coloring numbers for over half of the instances.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Structural sparsity, parameterized algorithms, parameterized complexity, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.44

Related Version *Based on Alexander Dobler's master thesis*: `https://doi.org/10.34726/hss.2021.91580` [11]
Full Version: `https://arxiv.org/abs/2203.03358` [13]

Supplementary Material *Software*: `https://doi.org/10.5281/zenodo.5732923` [12]

Funding *Alexander Dobler*: Supported by the Vienna Science and Technology Fund (WWTF) under grant ICT19-035.
Manuel Sorge: Supported by the Alexander von Humboldt Foundation.
Anaïs Villedieu: Supported by the Austrian Science Fund (FWF) under grant P31119.

Acknowledgements We thank Nadara et al. [32] for publishing their code, some small parts of which we reused in our implementations.

1 Introduction

A *degeneracy ordering* of a graph G can be obtained by iteratively removing an arbitrary vertex of minimum degree from G and putting it in front of the current ordering [30]. The *degeneracy* of a graph is the largest degree of a vertex encountered at removal. Degeneracy orderings are immensely useful when solving various tasks on graphs both in theory and practice [8, 18, 27, 37, 22, 6]. A key observation is that many graphs in practice have small degeneracy (e.g., in the order of hundreds for millions of edges) [18]. Thus, when looking for, e.g., maximum-size cliques, it is sufficient to look within the small number of neighbors in front of each vertex in the degeneracy ordering.


© Alexander Dobler, Manuel Sorge, and Anaïs Villedieu;
licensed under Creative Commons License CC-BY 4.0

However, degeneracy is not robust under local changes: e.g., contracting edges in a graph may arbitrarily increase its degeneracy. This property makes problems intractable on graphs that have small degeneracy if these problems are less local than finding maximum-size cliques. For example, detecting mild clique relaxations is hard on graphs of bounded degeneracy [27, 23]. Hence, we are searching for robust sparsity measures within the framework of structural sparsity [33].

We can obtain a robust variant of the class of graphs with bounded degeneracy by using the family of measures called weak r-coloring numbers [26]. For an integer r, the weak r-coloring number (wcol_r) of a graph G is the least integer k such that there is a vertex ordering with the property that, for each vertex v, there are at most k vertices u that are reachable from v by a path P of length at most r such that P does not use vertices that come before u in the ordering. The integer r, also called radius, interpolates between the degeneracy plus one ($r = 1$) and the so-called treedepth of G ($r = |V(G)|$ [20]), a measure of tree-likeness [3, 4, 25] and target of a recent implementation challenge [28].

It is important to compute the wcol_r of real-world graphs for two reasons. First, the wcol_r plays an important role in algorithmic and combinatorial techniques in structural sparsity (e.g., [2, 7, 15, 16, 17, 24, 29, 34, 36, 40]). For instance, a central concept therein is nowhere denseness and a subgraph-closed class of graphs is nowhere dense if and only if for each fixed integer r and $\epsilon > 0$ each graph G in the class has wcol_r at most $O(|V(G)|^{\epsilon})$ [42]. Thus, obtaining the wcol_rs of real-world graphs will help us gauge how well this theory fits practice. Second, there is a prospect that wcol_rs will help us solve other computational problems on real-world graphs more efficiently: On nowhere-dense graph classes each problem expressible in first-order logic can be solved in near-linear time [21]. There is indeed indication that relevant classes of real-world networks, including certain scale-free networks, are nowhere dense [10]. Thus, wcol_rs may help us transfer the above theoretical near-linear-time algorithms into something practically useful. For example, this work is underway for counting subgraphs [35, 39], which is the underlying computational problem of computing graph motifs or graphlets in biological and social networks [31, 38].

Computing the wcol_r of a graph is an algorithmically challenging task: for each $r \geq 2$ it is NP-complete [20, 5]. So far, there is but one work that studies computing upper bounds for wcol_rs in real-world graphs: Nadara et al. [32] used greedy heuristics that build the associated vertex ordering by iteratively choosing a yet unordered vertex that seems favorable and putting it at the front (or the back) of the current subordering (an ordering of a subset of all vertices). Afterwards, they apply local-search techniques that make local shifts in the ordering that decrease the associated weak r-coloring number. This yields upper bounds; to date the true weak r-coloring numbers of the studied graphs are unknown, even for the smallest part of Nadara et al.'s dataset that contains graphs with 62 to 930 edges.

In this work, we study a paradigm that has previously been successfully applied to improve the quality of the results computed by greedy heuristics: turbocharging [41, 14, 1, 19]. The basic idea is that we pre-specify an upper bound $k \in \mathbb{N}$ on the wcol_r of the ordering that we want to compute. We carry out the greedy heuristic that computes the ordering iteratively. If at some point it can be detected that the ordering will yield wcol_r larger than k – the *point of regret* – we start a turbocharging algorithm. This algorithm tries to modify the current ordering, by reordering or replacing certain vertices, so as to make it possible to achieve wcol_r at most k again. Then we continue with the greedy heuristic.

Our contribution is to develop the turbocharging algorithms applied at the point of regret, obtaining running-time guarantees and lower bounds, and to implement, engineer, and evaluate these algorithms on Nadara et al.'s dataset. The two main approaches that we

study are as follows. We fix a *reconstruction parameter* $c \in \mathbb{N}$. At the point of regret, in order to obtain a subordering of wcol_r at most k we either (a) replace the last c vertices of the current subordering with new, yet unordered vertices or (b) take c vertices out of the ordering and merge them into the subordering at possibly different positions. The formal definitions are given in Section 2. We show that both approaches are NP-hard in general (see Section 3) and hence we also consider the influence of small parameters on the achievable running-time guarantees. That is, we aim to show *fixed-parameter tractability* by giving algorithms with $f(p) \cdot n^{O(1)}$ running time for a small parameter p and input size n. On the negative side, approach (a) is W[1]-hard with respect to even both c and k (Theorem 2). That is, an algorithm with running time $f(c,k) \cdot \mathrm{poly}(n)$ is unlikely, where n is the number of vertices. This stands in contrast to Gaspers et al. [19] who obtained such an algorithm when the goal is to compute the treewidth of the input graph instead of its wcol_r. On the positive side, approach (a) is trivially tractable in polynomial time for constant c. For approach (b), we obtain a fixed-parameter algorithm with respect to $c+k$ (Theorem 4). We implemented a set of algorithms including the two previously mentioned ones and report on implementation considerations and results in Section 4. The results indicate that on average the weak r-coloring numbers achieved by Nadara et al. [32] can be improved by 5 % by using our turbocharging algorithms. Using turbocharging we obtain smaller weak coloring numbers than all previous approaches on 181 of the in total 334 instances used by Nadara et al. [32].

Due to space constraints, we defer some details of proofs, implementations, and evaluations to a full version [13].

2 Preliminaries and turbocharging problems

General preliminaries. We only consider undirected, unweighted graphs G without loops. By $V(G)$ and $E(G)$ we denote the vertex set and edge set of G, respectively. For $S \subseteq V(G)$, $G[S]$ is the *induced subgraph* on vertices in S. A *path* $P = (v_1, \ldots, v_n)$ in G is a non-empty sequence of vertices, such that consecutive vertices are connected by an edge. The *length* of a path is $|V(P)| - 1$. In particular, a path of length 0 consists of a single vertex. We use $\mathrm{dist}_G(u,v)$ to denote the length of the shortest path between vertices u and v in G.

A *vertex ordering* L of a graph G is a linear ordering of $V(G)$. We write $u \prec_L v$ if vertex u precedes vertex v in L. In this case we say that u is *left of* v w.r.t. L. Equivalently, we write $u \preceq v$ if $u \prec v$ or $u = v$. We also denote vertex orderings L as sequences of its elements, that means $L = (v_1, \ldots, v_n)$ represents the vertex ordering where $v_i \prec_L v_j$ iff. $i < j$. We denote by $\Pi(G)$ the set of all vertex orderings of G. A *subordering* is a linear ordering of a subset $S \subseteq V(G)$. The notation L_S shall always denote a subordering where S is the set of vertices ordered in the subordering. We call the vertices in $V(G) \setminus S$ *free* with respect to L_S. Usually we will denote the set of free vertices with respect to a subordering by T. For a subordering L_S and $S' \subseteq S$ we denote by $L_S[S']$ the subordering that *agrees with* L_S on S', that is, for all $u, v \in S'$ we have that $u \prec_{L_S[S']} v$ iff. $u \prec_{L_S} v$, and all vertices in $V(G) \setminus S'$ are free w.r.t. $L_S[S']$. For a subordering L_S and $S' \supseteq S$, a subordering $L_{S'}$ is a *right extension* of L_S if $L_{S'}[S] = L_S$ and $u \prec_{L_{S'}} v$ for all $u \in S$ and $v \in S' \setminus S$. If $S' = V$, then $L_{S'}$ is called *full right extension*.

Weak coloring numbers. For a vertex ordering L of G and $r \in \mathbb{N}$, a vertex u is *weakly r-reachable* from a vertex v w.r.t. L if there exists a path P of length ℓ with $0 \leq \ell \leq r$ between u and v such that $u \preceq_L w$ for all $w \in V(P)$. Let $\mathrm{Wreach}_r(G, L, v)$ be the set of vertices that are weakly r-reachable from v in G w.r.t. L. The *weak r-coloring number* $\mathrm{wcol}_r(G, L)$ of a vertex ordering L is $\mathrm{wcol}_r(G, L) = \max_{v \in V(G)} |\mathrm{Wreach}_r(G, L, v)|$, and the weak r-coloring number $\mathrm{wcol}_r(G)$ of G is $\mathrm{wcol}_r(G) = \min_{L \in \Pi} \mathrm{wcol}_r(G, L)$.

Turbocharging. Our goal is to find a vertex ordering L of a given graph G with small $\mathrm{wcol}_r(G, L)$ by applying turbocharging. We start with two well-known iterative greedy heuristics (descriptions will follow) of Nadara et al. [32] that build vertex orderings with small weak r-coloring number from left to right. That is, these heuristics start with the empty subordering $L_S = \emptyset$ and in each step compute a right extension of L_S that contains one more vertex. This process is continued until the constructed subordering contains all vertices. A key observation about this process that we can use for turbocharging is that the size of the weakly reachable set of each vertex cannot decrease:

▶ **Observation 1.** *Let L_S be a subordering, $u, v \in V(G)$, and L be a full right extension of L_S. If $u = v$, or $u \in S$ and there exists a path P of length ℓ with $0 \leq \ell \leq r$ between u and v such that $u \preceq_{L_S} w$ for all $w \in V(P) \cap S$, then $u \in \mathrm{Wreach}_r(G, L, v)$.*

For u and v as in Observation 1 we extend the definition of weak r-reachability to suborderings L_S by defining $u \in \mathrm{Wreach}_r(G, L_S, v)$ and $\mathrm{wcol}_r(G, L_S) = \max_{v \in V(G)} |\mathrm{Wreach}_r(G, L_S, v)|$. We immediately obtain that $\mathrm{wcol}_r(G, L_S)$ is a lower bound; that is, if L is a full right extension of L_S (such as obtained by one of the heuristics), then $\mathrm{wcol}_r(G, L) \geq \mathrm{wcol}_r(G, L_S)$.

The two heuristics of Nadara et al. that we apply are:

- The Degree-Heuristic: This heuristic orders vertices by descending degree, ties are broken arbitrarily.
- The Wreach-Heuristic: For a subordering L_S, this heuristic picks the free vertex v with the largest $\mathrm{Wreach}_r(G, L_S, v)$. Ties are broken by descending degree.

Nadara et al. proposed several other heuristics, but those heuristics do not build vertex orderings from left to right, but in different orders. Additionally, the above heuristics are among the best-performing ones with regard to computed weak coloring numbers and runtime.

In what follows, let us assume that we want to compute a vertex ordering L of a graph G with $\mathrm{wcol}_r(G, L) \leq k$ where $k \in \mathbb{N}$. We might apply one of the heuristics until we obtain a subordering L_S such that $\mathrm{wcol}_r(G, L_S) > k$. We call such a subordering *non-extendable* (and otherwise the subordering is *extendable*); we also say that this point in the execution of the heuristic is the *point of regret*. We then consider two exact *turbocharging problems* that try to locally augment L_S, such that it is extendable again. If the *turbocharging algorithms* for these problems that we later propose are successful in making L_S extendable, then we continue applying the heuristic until we have to repeat this process (trying to find a vertex order L with $\mathrm{wcol}_r(G, L) \leq k$).

Motivated by a turbocharging algorithm for computing tree-decompositions by Gaspers et al. [19] we consider replacing a bounded-length suffix of the current subordering. That is, we specify a *reconstruction parameter* $c \in \mathbb{N}$ in advance and, at the point of regret, we remove the last c vertices from L_S and then try to add c (possibly) different vertices. This leads to the following turbocharging problem.

Incremental Conservative Weak r-coloring (IC-WCOL(r))
Instance: A graph G, a subordering L_S, and positive integers k and c.
Question: Is there an extendable right extension $L_{S'}$ of L_S such that $|S' \setminus S| = c$?

Our second turbocharging approach is based on a vertex v with $|\mathrm{Wreach}_r(G, L_S, v)| > k$. Therein, instead of the suffix of the current order, we choose a set S_2 of vertices related to the weakly r-reachable set of v (details follow in Section 4). We remove the vertices in S_2 from L_S, leaving us with the subordering L_{S_1}, and then we try to reinsert the vertices in S_2 while decreasing the weak coloring number.

WCOL-Merge(r)
Instance: A graph G, an integer k, two disjoint sets S_1 and S_2 such that $S_1, S_2 \subseteq V(G)$, and a subordering L_{S_1}.
Question: Is there an extendable subordering $L_{S_1 \cup S_2}$ such that $L_{S_1 \cup S_2}[S_1] = L_{S_1}$?

Herein, we put the *reconstruction parameter* c equal to $|S_2|$ and denote by T the set of free vertices $V(G) \setminus (S_1 \cup S_2)$.

3 Algorithms and running-time bounds

We continue by providing algorithmic upper and lower bounds for Incremental Conservative Weak r-coloring (IC-WCOL(r)) and WCOL-Merge(r), starting with IC-WCOL(r).

3.1 IC-WCOL(r)

The first theoretical result that we want to present is the NP-hardness of Incremental Conservative Weak r-coloring for each $r \geq 1$ by giving a reduction from Independent Set. Independent Set takes as input a graph G and a positive integer p and asks if there is a set of vertices I of size at least p such that $\{u, v\} \not\subseteq I$ for all $\{u, v\} \in E(G)$.

▶ **Theorem 2.** *For each fixed $r \geq 1$,* Incremental Conservative Weak r-coloring *is NP-hard and $W[1]$-hard when parameterized by $k + c$.*

The proof can be found in the full version of this paper [13] and gives a polynomial reduction from of Independent Set which is NP-complete to an instance of IC-WCOL(r). The idea is that the parameter p – the desired independent set size of a given Independent Set instance – is transformed to the parameters $c = p$ and $k = 2$ of IC-WCOL(r), giving us a parameterized reduction from Independent Set to IC-WCOL(r). Independent Set is W[1]-hard when parameterized by p, and thus we obtain the stated $W[1]$-hardness.

On the other hand, it is not hard to see that Incremental Conservative Weak r-coloring is in XP: We can simply try placing any of the vertices from $V(G) \setminus S$ into the next free position right of L_S. As there are c free positions the overall algorithm runs in $\mathcal{O}(|V(G) \setminus S|^c \cdot |V(G)|^{\mathcal{O}(1)}) \subseteq \mathcal{O}(|V(G)|^c \cdot |V(G)|^{\mathcal{O}(1)})$ time.

▶ **Proposition 3.** Incremental Conservative Weak r-coloring *parameterized by the reconstruction parameter c is in XP.*

Our algorithm for Incremental Conservative Weak r-coloring is based on Proposition 3, we will go into more detail in Section 4.

3.2 WCOL-Merge(r)

It is easy to see that WCOL-Merge(r) is NP-hard for $r \geq 2$: Given a graph G, we can decide $\mathrm{wcol}_r(G) \leq k$ by creating an instance (H, S_1, S_2, L_{S_1}) of WCOL-Merge(r), setting $S_1 = L_{S_1} = \emptyset$, $H = G$, and $S_2 = V(G)$. As deciding $\mathrm{wcol}_r(G) \leq k$ is NP-hard for $r \geq 2$ [20, 5], so is WCOL-Merge(r). On the positive side, we can show fixed-parameter tractability of WCOL-Merge(r) parameterized by k and $|S_2|$.

▶ **Theorem 4.** WCOL-Merge(r) *is solvable in time $\mathcal{O}(|S_2|! \cdot k^{|S_2|} \cdot |V(G)|^{\mathcal{O}(1)})$. In particular,* WCOL-Merge(r) *is fixed-parameter tractable when parameterized by $k + |S_2|$.*

Intuitively, the algorithm behind Theorem 4 tries to place each vertex v in S_2 one by one by trying all relevant positions. The key insight is that only few positions are relevant (called breakpoints below). Namely those positions that correspond to vertices that are weakly r-reachable from v when placed at the end of the ordering. As only few vertices can be reachable from v when placed in the correct position, we only need to try the first k corresponding positions.

To describe the algorithm we need definitions for two operations that we use throughout.

▶ **Definition 5.** *Let G be a graph, $L_S = (s_1, \dots, s_n)$ a subordering, and $v \in V(G) \setminus S$. We denote by* $\text{placeafter}(L_S, s_i, v)$ *the subordering of vertices $S \cup \{v\}$ that is obtained by placing v directly after s_i. To be precise,* $\text{placeafter}(L_S, s_i, v) := (s_1, \dots, s_i, v, s_{i+1}, \dots, s_n)$. *Equivalently,* $\text{placebefore}(L_S, s_i, v) := (s_1, \dots, s_{i-1}, v, s_i, \dots, s_n)$.

This leads to the definitions of breakpoints, which are crucial for the proof of Theorem 4.

▶ **Definition 6.** *Let G be a graph, $L_S = (s_1, \dots, s_n)$ a subordering, and $v \in V(G) \setminus S$. A vertex $s \in S$ is called* breakpoint *of v if* $\text{Wreach}_r(G, \text{placebefore}(L_S, s, v), v) \neq \text{Wreach}_r(G, \text{placeafter}(L_S, s, v), v)$. *Let* $\text{bp}(G, L_S, v) \subseteq S$ *be the set of breakpoints of v.*

We also notice another useful property of breakpoints, which is proved in the full version of this paper [13].

▶ **Lemma 7.** *Let $v \in V(G) \setminus S$. We have $s \notin \text{bp}(G, L_S, v)$ if and only if for all $u \in V(G)$*

$$\text{Wreach}_r(G, \text{placebefore}(L_S, s, v), u) = \text{Wreach}_r(G, \text{placeafter}(L_S, s, v), u).$$

If we add a vertex v to a subordering L_S to obtain a new subordering $L_{S\cup\{v\}}$, then the weakly reachable vertices $\text{Wreach}_r(G, L_{S\cup\{v\}}, v)$ consist of v and a subset of $\text{bp}(G, L_S, v)$. We formalize this as follows.

▶ **Lemma 8.** *Let G be a graph, $L_S = (s_1, \dots, s_n)$ be a subordering, and $v \in V(G) \setminus S$. Furthermore, let $L_{S\cup\{v\}}$ be a subordering such that $L_{S\cup\{v\}}[S] = L$. Then*

$$\text{Wreach}_r(G, L_{S\cup\{v\}}, v) \setminus \{v\} = \{s \in \text{bp}(G, L_S, v) \mid s \preceq_{L_{S\cup\{v\}}} v\}.$$

Proof. Let X be the set $\{s \in \text{bp}(G, L_S, v) \mid s \preceq_{L_{S\cup\{v\}}} v\}$, we prove both inclusions of the equation $\text{Wreach}_r(G, L_{S\cup\{v\}}, v) \setminus \{v\} = X$.

Assume that $s \in (\text{Wreach}_r(G, L_{S\cup\{v\}}, v) \setminus \{v\})$. As s is weakly r-reachable from v, there is a path $P = (v, u_1, \dots, u_\ell, s)$ of length of at most r that does not go left of s w.r.t. to $L_{S\cup\{v\}}$. Consider the same path P in $\text{placeafter}(L_S, s, v)$. Clearly, s is also weakly r-reachable in this subordering because of the same path. Contrary to that, s cannot be weakly r-reachable from v in $\text{placebefore}(L_S, s, v)$ because v is left of s in that subordering. Hence, $s \in \text{bp}(G, L_S, v)$ and $\text{Wreach}_r(G, L_{S\cup\{v\}}, v) \setminus \{v\} \subseteq X$.

Assume that $s \in X$. Then s must be weakly r-reachable from v w.r.t. $\text{placeafter}(L_S, s, v)$ through a path P of length at most r. But s is also weakly r-reachable from v w.r.t. $L_{S\cup\{v\}}$ through the same path P. Hence, $X \subseteq \text{Wreach}_r(G, L_{S\cup\{v\}}, v) \setminus \{v\}$ also holds. ◀

Using the above tools, we can now formally describe Algorithm 1, which obtains the stated runtime of Theorem 4 and is given as a recursive function RECURSIVE-MERGE. As alluded to before, the intuition is that for each vertex $v \in S_2$ we only have to consider placing it before its breakpoints w.r.t. L_{S_1}. As the breakpoints of a vertex will be in its weakly r-reachable set, only the leftmost k breakpoints are relevant. A detailed proof of the correctness and the runtime can be found in the full version of this paper [13].

Algorithm 1 Recursive FPT-algorithm for WCOL-MERGE(r).

```
1  Recursive-merge(S_1, S_2, T, L_{S_1}):
2    if |S_2| = 0 ∧ ∀v ∈ V(G) : |Wreach_r(G, L_{S_1}, v)| ≤ k then return L_{S_1} ;
3    for v ∈ S_2 do
4      for s ∈ bp(G[S_1 ∪ T ∪ {v}], L_{S_1}, v) do
5        L_{S_1∪{v}} ← placebefore(L_{S_1}, s, v);
6        if |Wreach_r(G[S_1 ∪ T ∪ {v}], L_{S_1∪{v}}, v)| ≤ k then
7          answer ← Recursive-merge(S_1 ∪ {v}, S_2 \ {v}, T, L_{S_1∪{v}});
8          if answer ≠ false then return answer;
9        end
10     end
11     s ←rightmost vertex of S_1 w.r.t. L_{S_1};
12     L_{S_1∪{v}} ← placeafter(L_{S_1}, s, v);
13     if |Wreach_r(G[S_1 ∪ T ∪ {v}], L_{S_1∪{v}}, v)| ≤ k then
14       answer ← Recursive-merge(S_1 ∪ {v}, S_2 \ {v}, T, L_{S_1∪{v}});
15       if answer ≠ false then return answer;
16     end
17   end
18   return false
```

4 Implementation and experiments

This section contains implementation details for the heuristics and both turbocharging algorithms. Furthermore, we give the experimental setup and experimental results.

4.1 Algorithm implementations and heuristic improvements

Interesting details are omitted from the algorithmic results of Section 3. We want to give some implementation details for IC-WCOL(r) and WCOL-MERGE(r), and how the algorithms for these problems are combined with the heuristics. Additionally, we give a third turbocharging approach called IC-WCOL-RL(r) in this section.

In our implementations of heuristics and turbocharging algorithms we store and update the current subordering L_S in a simple array. We also store and update the sets $\mathrm{Wreach}_r(G, L_S, v)$ and $\mathrm{Wreach}_r^{-1}(G, L_S, v) = \{w \in V(G) : v \in \mathrm{Wreach}_r(G, L_S, w)\}$ (see below for their usage). Updating weakly r-reachable sets during placements and removals of vertices v is done by computing the set $\mathrm{Wreach}_r^{-1}(G, L_S, v)$ via a breadth-first search that respects the order L_S, and updating the corresponding weakly r-reachable sets.

Additionally, we slightly adapt the Degree-Heuristic: We aim for vertex orderings L with $\mathrm{wcol}_r(G, L) \leq k$. Consider a subordering L_S that was created by the heuristic and needs to be extended. To obtain weak coloring number k it would intuitively make sense to place a free vertex v with $\mathrm{wcol}_r(G, L_S) = k$ immediately to the right of that subordering s.t. its weakly r-reachable set cannot increase anymore. This is indeed always correct – if there is a full right extension L of L_S with $\mathrm{wcol}_r(G, L) \leq k$, then there is another one that starts by placing v immediately to the right of L_S (for a formal statement and a proof we refer to the full version of this paper [13]). In our implementation of the Degree-Heuristic we apply this observation and place such a vertex v immediately. The Wreach-Heuristic does this implicitly.

We continue by explaining individual details for IC-WCOL(r) and WCOL-Merge(r), and how they are applied to a non-extendable subordering L_S with free vertices T.

IC-WCOL(r). We have implemented the XP-algorithm for Incremental Conservative Weak r-coloring as outlined in Proposition 3. Given a subordering L_S, we have to extend L_S to the right by c vertices, that means that we have c positions to fill. We implement a search tree algorithm that fills these positions from left to right recursively. That is, in a search tree node we try all possibilities of placing a free vertex into the leftmost free position i and recurse into search tree nodes that try placing the remaining free vertices into position $i+1$, and so on, until all c positions are filled.

If after a placement of a vertex we obtain a non-extendable subordering, we can cut off this branch of the search tree, as weakly r-reachable sets of vertices can only increase in this branch. We also store edges of $G[T]$ separately as an array of hash sets. This enables, in a search tree node, to update the sets $\mathrm{Wreach}_r^{-1}(G, L_S, v)$ on placement/removal by a simple depth-r breadth-first-search in $G[T]$. This decreases the number of enumerated edges compared to the trivial approach.

Free vertices T are placed into a position i in a specific order inside a search tree node: let L_S be the non-extendable subordering that triggered turbocharging and let v be the rightmost vertex of L_S. We try placing $u_1 \in T$ before $u_2 \in T$ into the free slot i if $\mathrm{dist}_G(u_1, v) < \mathrm{dist}_G(u_2, v)$. Preliminary experiments suggested that this order is preferable to a random one. We compute $\mathrm{dist}_G(u, v)$ for all u, v with Johnson's Algorithm [9] for sparse graphs once in the beginning.

WCOL-Merge(r). We apply WCOL-Merge(r) to a non-extendable subordering L_S in the following way. Let $U = \{v \in V(G) : |\mathrm{Wreach}_r(G, L_S, v)| > k\}$ be the set of *overfull* vertices. Let c be a positive integer and let X be a random subset of $\bigcup_{v \in U} \mathrm{Wreach}_r(G, L_S, v)$ of size $\min(c, |\bigcup_{v \in U} \mathrm{Wreach}_r(G, L_S, v)|)$. If the size of X is less than c, we randomly add additional vertices from $V(G)$ to X, until the size of X is c. We try to fix L_S by defining an instance of WCOL-Merge(r). Namely, we set $S_1 = S \setminus X$, $S_2 = X$, and $L_{S_1} = L_S[S_1]$. We then solve this instance using Algorithm 1. By Theorem 4 we obtain a turbocharging algorithm that has fixed-parameter tractable running time when parameterized by the desired coloring number k and reconstruction parameter c.

In our implementation we apply WCOL-Merge(r) multiple times with different randomly selected sets X as defined above, until we obtain an extendable subordering. Preliminary experiments showed that choosing the whole set $\bigcup_{v \in U} \mathrm{Wreach}_r(G, L_S, v)$ as X leads to timeouts often whereas random subsets still allowed us to fix L_S. If we do not find an extendable subordering after the 10th application of WCOL-Merge(r), we report that turbocharging was not successful.

We now discuss the implementation of Algorithm 1. Consider the vertex v in Algorithm 1. We only have to iterate over the k leftmost breakpoints of v due to Lemma 8, which can be easily done by storing and maintaining $\mathrm{Wreach}_r^{-1}(G[S_1 \cup T \cup \{v\}], L_{S_1 \cup T}, v)$. Let s be a breakpoint of v and let $L_{S_1 \cup \{v\}} = \mathrm{placeafter}(L_{S_1}, s, v)$. The leftmost $s' \in \mathrm{Wreach}_r^{-1}(G[S_1 \cup T \cup \{v\}], L_{S_1 \cup \{v\}}, v)$ that is not v is the next possible breakpoint of v. Additionally, we do not need to recurse if the size of some set $\mathrm{Wreach}_r(G[S_1 \cup T], L_{S_1}, v)$ exceeds k for some $v \in S_1 \cup T$, as these sets can only increase in subsequent recursion calls.

We also know that subsets of some weakly r-reachable sets of vertices T are already fixed. Namely, for all vertices v in the r-neighborhood of u in $G[T \cup \{u\}]$ with $u \in S_2$, vertex u will always be in the weakly r-reachable set of v if u is placed somewhere into the subordering L_{S_1}. We take this into account when calculating lower bounds for the sizes of weakly r-reachable sets of vertices in T (and break the search if they exceed size k).

IC-WCOL-RL(r). We also implemented a turbocharging algorithm that is not discussed above. It is based on the *Sreach-Heuristic*, which builds a vertex ordering of low weak coloring number from right to left (instead of from left to right as above) [32]. It starts with an empty subordering L_S and, during each step, the heuristic adds to the left front of L_S a free vertex v that minimizes the number of so-called potentially strongly r-reachable vertices after being placed. Herein, a vertex u is *potentially strongly r-reachable* w.r.t. L_S from a vertex $v \in S$ if $u \in \mathrm{Wreach}_r(G[S], L_S, v)$ or there is a path P of length at most r from v to u in G such that $V(P) \cap T = \{u\}$. It can be shown (refer to a formal statement and a proof in the full version of this paper [13]) that the set of potentially strongly r-reachable vertices of v only grows when extending L_S to the left and that, when $S = V(G)$, this set equals $\mathrm{Wreach}_r(G, L_S, v)$. Thus, we define a *point of regret* for this heuristic as a point in the execution where there is a vertex v such that the size of its potentially strongly r-reachable set exceeds the desired weak coloring number k. Accordingly, we say that L_S is *non-extendable*, and otherwise it is *extendable*. We then solve the turbocharging problem in which we aim to replace the c leftmost vertices of L_S with arbitrary free vertices in order to make L_S extendable again. We do this using a search-tree algorithm analogous to IC-WCOL(r). We also use the above turbocharging approach with a heuristic that chooses the next vertex among the free vertices based on the smallest degree. We call this heuristic Degree-Heuristic[1].

We tried further heuristic optimizations, mainly for the left-to-right approaches, based on lower bounds (for early search termination), guided branching (towards faster decomposition into trivial instances), and ordered adjacency lists (to speed up computation of weakly reachable sets) but they did not improve the resulting coloring numbers.

4.2 Experiments setup

Computation environment. All experiments were performed on a cluster of 20 nodes. Each node is equipped with two Intel Xeon E5-2640 v4, 2.40GHz 10-core processors and 160 GB RAM. The optimization for each instance and an algorithm was pinned to a specific core of a cluster node (simultaneous multithreading was disabled). All implementations of heuristics and turbocharging algorithms were done in C++17, and made use of the Boost library[2], version 1.77.0. The code was compiled on Linux with `g++` version 7.5.0 and with the flags `-std=c++17 -O2`. The optimization process (see *application of turbocharging* below) that calls the heuristics combined with the turbocharging algorithms (implemented in C++) was implemented in Python3 and executed with Python 3.7.13. A memory limit of 16 GB was set (a process only starts if the required memory is free). The source code is available online [12].

Instances. Each instance in our data set is a tuple consisting of a graph G and a radius $r \in \mathbb{N}$. The radii r are between 2 and 5, as also used by Nadara et al. [32]. The graphs G form a subset of the graphs used by Nadara et al. This enables us to use weak coloring numbers of orderings computed by Nadara et al. as a baseline. Furthermore, we can compare for a heuristic, the improvement achieved by the local search of Nadara et al. to the improvement achieved by our turbocharging algorithms.

The graphs consist of real-world data, the PACE 2016 Feedback Vertex Set problems, random planar graphs, and random graphs with bounded expansion. Nadara et al. classified the graphs into four classes based on the number of edges – small (up to 1k edges), medium (up to 10k edges), big (up to 48k edges), and huge. For a detailed explanation and references

[1] The name is the same as in the left-to-right setting; there will be no confusion between the two because the direction will be clear from the context.
[2] `https://www.boost.org/`

Algorithm 2 Algorithm for iteratively decreasing the weak coloring number by turbocharging.

Input: A graph G, an integer r, a heuristic $\mathcal{H}$, and a turbocharged heuristic $TC\text{-}\mathcal{H}$
Output: An ordering L of vertices V

```
L ← ordering of vertices V computed by the heuristic H;
k ← wcol_r(G, L);
Start a timer; after t seconds abort the program, and return the current value of L
while true do
    c ← 1;
    while true do
        Try to compute an ordering of vertices V with weak r-coloring number k − 1
          using TC-H with reconstruction parameter c;
        If successful, assign this ordering to L, set k ← wcol_r(G, L), and break;
        Otherwise, set c ← c + 1;
    end
end
```

for all input graphs we refer to Nadara et al. [32]; the instances are available online[3]. We considered all instances except those where one of the three heuristics of Nadara et al. that we also consider did not yield a result (the Degree-Heuristic, Wreach-Heuristic, and Sreach-Heuristic). That is, they timed out after 300 seconds or ran into a memory limit (16 GB). In total, our dataset contains 334 instances.

Application of turbocharging. Our application of turbocharging with a heuristic $\mathcal{H}$ works as follows. Given a graph G and a radius r, we start with a run of $\mathcal{H}$ without turbocharging to produce a vertex ordering for G with a baseline weak r-coloring number k. We then start a timer that runs for t seconds, aborting the rest of the algorithm when it terminates. We then iteratively decrease k and apply $\mathcal{H}$ together with the turbocharging approach to try and find a vertex ordering for G with weak r-coloring number at most k. In each such try, we start with the reconstruction parameter $c = 1$. If no ordering with the desired weak r-coloring number was produced by the turbocharged heuristic, we increase c by one and try again. If an ordering with weak r-coloring number $k' \leq k$ was produced, we set $k = k' - 1$ and repeat the process. The precise algorithm is given in Algorithm 2.

For our experiments we applied all compatible combinations of heuristics and the turbocharged versions to each instance. Furthermore, we computed orderings for radii ranging from 2 to 5, motivated by Nadara et al. who used the same values. We run all experiments twice, once with timeout $t = 300s$ and once with $t = 3600s$. Results with $t = 300s$ are directly compared with the results of Nadara et al. who used 300 seconds as timeout. Results with $t = 3600s$ give us the ability to investigate the potential of turbocharging over longer periods of time.

[3] `https://kernelization-experiments.mimuw.edu.pl/`

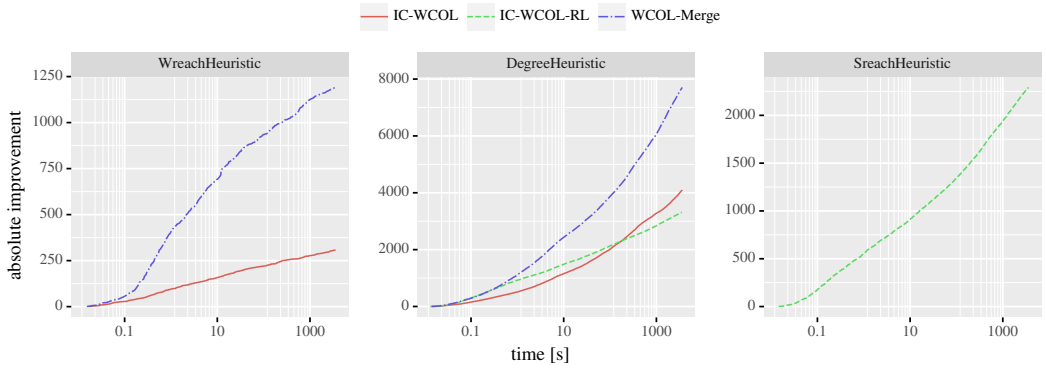

Figure 1 Line plot of the cumulative absolute improvements over time achieved by the turbocharging approaches broken down by the underlying heuristics. The x-axis (time) is scaled logarithmically.

4.3 Results

We now show to which extent our turbocharging approaches improve the results of heuristics, compare the achieved weak coloring numbers to the ones of Nadara et al., and provide observations about the performance of turbocharging.

Impact of turbocharging. Turbocharging has the advantage that investing gradually more time will yield gradually better results – setting the reconstruction parameter to $c = |V(G)|$, we (in theory) even can provably obtain the optimum. Figure 1 shows the cumulative sum of absolute improvements over time when comparing each turbocharged heuristic with the underlying plain heuristic. That is, for a specific time t, the cumulative sum of absolute improvements for a turbocharging algorithm $\mathcal{A}$ and a heuristic $\mathcal{H}$ is $\sum_{I \in \text{instances}}(k_{I,\mathcal{H}} - k_{I,\mathcal{A},\mathcal{H},t})$, where $k_{I,\mathcal{H}}$ is the coloring number achieved by the heuristic and $k_{I,\mathcal{A},\mathcal{H},t}$ is the coloring number achieved by the turbocharged heuristic after time t. Note that the y-axes do not have the same ranges as the underlying heuristics have different performance levels. All plots exhibit logarithmic or similar to logarithmic growth, which means that the gained absolute improvement is approximately logarithmic in the invested time in the most cases. WCOL-Merge(r) clearly yields faster and larger improvements than IC-WCOL(r), and it also supersedes IC-WCOL-RL(r) for the Degree-Heuristic. One reason may be that WCOL-Merge(r) is fixed-parameter tractable and the associated parameters are small.

Further evaluations of the executions of turbocharging algorithms are analysed in detail in the full version of this paper [13]. Some observations therein are that the number of applications of turbocharging per run of a heuristic ranges in the order of at most hundreds for IC-WCOL(r) and WCOL-Merge(r) and on average in the single digits; for IC-WCOL-RL(r) these numbers are one to two orders of magnitude larger. Successful applications of a turbocharged heuristic (those where the weak coloring number could be improved) mostly only use very little time and search tree nodes, and have small reconstruction parameters – mostly $c = 1$. That is, it is mostly the case that a heuristic wants to place a vertex that is "suboptimal", while placing nearly any other vertex will achieve lower weak coloring number. The fraction of time spent on turbocharging is also low, which means that most of the time when we can improve the weak coloring number achieved by a heuristic, this can be done easily and in little time.

Table 1 IC-WCOL(r): White columns give relative improvements, light gray columns give quality ratios, dark gray columns give average/maximum absolute improvements. For improvements, we compare to the underlying heuristic without turbocharging and without local search. Time limit: 300 s.

tests	r	Wreach IC-WCOL(r)			Degree IC-WCOL(r)		
small	2	-6.2%	0.8/5	7.2%	-7.3%	3.2/14	18.4%
	3	-7.3%	1.0/6		-9.3%	4.3/20	
	4	-11.3%	1.7/7		-11.2%	4.2/15	
	5	-15.8%	1.9/8		-16.8%	3.9/12	
medium	2	-5.6%	0.6/2	3.7%	-7.1%	5.0/14	17.4%
	3	-9.2%	1.0/11		-9.7%	9.0/35	
	4	-11.6%	0.3/2		-15.8%	9.9/31	
	5	-16.8%	1.3/15		-20.6%	9.9/41	
big	2	-9.6%	0.1/1	0.7%	-21.6%	7.2/31	12.0%
	3	-9.5%	0.4/3		-20.8%	15.3/47	
	4	-12.7%	0.3/2		-32.5%	14.5/42	
	5	-38.3%	0.2/1		-30.4%	13.7/38	
huge	2	-2.0%	0.1/1	0.2%	-39.1%	2.8/16	0.9%
	3	-21.4%	0.1/1		-35.0%	1.9/6	
	4	-6.9%	0.0/0		-29.9%	1.8/5	
	5	-8.9%	0.3/1		-16.5%	1.7/4	

By dataset group and radius. Next, we fix a time threshold of 300s (same as Nadara et al.) that we might reasonably invest in practice for computing weak coloring numbers. We present the improvements in weak coloring numbers gained by turbocharging over the plain heuristics broken down according to the instance group (small, medium, big, huge) and the radius r. We again provide absolute improvements when comparing with the underlying heuristic, and we also consider the average *relative improvement* of the weak r-coloring number when comparing the turbocharged heuristic to the plain heuristic; that is, the relative improvement is $1 - k_{I,\mathcal{A},\mathcal{H},t}/k_{I,\mathcal{H}}$ for $t = 300s$. For each turbocharging algorithm we show results for both turbocharged heuristics. For IC-WCOL(r) and WCOL-Merge(r) these are the Wreach- and Degree-Heuristic, and for IC-WCOL-RL(r) these are the Sreach- and Degree-Heuristic. The results are given in three tables corresponding to the three turbocharging approaches.

We furthermore compare the achieved coloring numbers of each approach to the best coloring numbers computed by Nadara et al.: For each instance I, let $\mathsf{bestNadara}(I)$ be the smallest weak r-coloring number of an ordering of vertices of instance I achieved by an approach of Nadara et al. Note that they implemented seven different heuristics and for each computed ordering they applied a local search to iteratively reduce the weak r-coloring number. To evaluate one of our approaches, we take the weak r-coloring number $k_{I,\mathcal{A},\mathcal{H},t}$ for instance I obtained by our approach for $t = 300s$ and compute the average $1 - k_{I,\mathcal{A},\mathcal{H},t}/\mathsf{bestNadara}(I)$ (in percent) taken over all instances I in the corresponding data set. We call this value *quality ratio*. Note that positive values mean that the approach achieves lower weak coloring numbers on average when compared to the best weak coloring numbers achieved by Nadara et al.

Table 2 WCOL-MERGE(r): White columns give relative improvements, light gray columns give quality ratios, dark gray columns give average/maximum absolute improvements. For improvements, we compare to the underlying heuristic without turbocharging and without local search. Time limit: 300 s.

tests	r	Wreach WCOL-MERGE(r)			Degree WCOL-MERGE(r)		
small	2	-1.0%	1.4/5	19.1%	0.3%	4.1/11	33.7%
	3	3.3%	2.8/8		2.9%	6.3/25	
	4	0.7%	4.2/11		2.3%	7.2/23	
	5	-1.2%	5.4/14		-0.2%	8.0/19	
medium	2	2.4%	1.6/7	15.0%	1.1%	6.4/16	32.7%
	3	0.4%	2.7/15		0.6%	11.6/46	
	4	-0.2%	2.9/13		-1.9%	13.3/36	
	5	-5.3%	4.2/16		-5.5%	15.5/50	
big	2	-0.3%	1.7/7	9.4%	-7.5%	11.0/32	24.2%
	3	-1.8%	2.1/9		-8.1%	23.4/65	
	4	-4.2%	2.8/11		-19.3%	26.6/63	
	5	-27.2%	4.9/26		-20.5%	26.6/67	
huge	2	-0.6%	1.7/5	1.0%	-14.9%	28.4/84	12.2%
	3	-20.5%	1.8/5		-21.9%	27.4/49	
	4	-6.7%	0.5/2		-27.6%	11.2/20	
	5	-8.7%	1.0/2		-15.5%	6.0/14	

In Table 1 we present the performance of the IC-WCOL(r) approach. It is evident that the relative and absolute improvements achieved for the Degree-Heuristic is significantly higher than for the Wreach-Heuristic, although this is partly due to the fact that the Degree-Heuristic achieves worse results than the Wreach-Heuristic before turbocharging. Relative and absolute improvements decrease for larger instances.

Table 2 contains the results for WCOL-MERGE(r). Here, turbocharging achieves positive quality ratios for some instance classes and radii. The relative and absolute improvements are much larger than for IC-WCOL(r), especially for the huge instances and the Degree-Heuristic. It is also interesting that while the Degree-Heuristic generally computes orderings of higher weak coloring number than the Wreach-Heuristic, the turbocharged version of the Degree-Heuristic computes orderings of similar or even lower weak coloring numbers than the turbocharged version of the Wreach-Heuristic for the small and medium instances. We do not see an obvious reason for that, but it could again indicate the power of the fixed-parameter algorithm.

Table 3 contains the results for IC-WCOL-RL(r). Although the relative and absolute improvements of turbocharging the Degree-Heuristic are slightly higher, the quality ratios for the turbocharged version of the Sreach-Heuristic are significantly better. This could imply that IC-WCOL-RL(r) struggles to turbocharge slightly worse heuristics such as the Degree-Heuristic. Furthermore, we see that for the Sreach-Heuristic the quality ratios are better for larger radii. The reason for this could be that the Sreach-Heuristic performs well for larger radii even before turbocharging. We also notice that for the medium graph class the quality ratios get worse. The reason is that the implementation of IC-WCOL-RL(r) is slightly more computationally expensive than for the other approaches.

Table 3 IC-WCOL-RL(r): White columns give relative improvements, light gray columns give quality ratios, dark gray columns give average/maximum absolute improvements. For improvements, we compare to the underlying heuristic without turbocharging and without local search. Time limit: 300 s.

tests	r	Sreach IC-WCOL-RL(r)			Degree IC-WCOL-RL(r)		
small	2	-2.1%	3.2/28	15.9%	-7.5%	3.1/11	22.0%
	3	1.6%	2.7/10		-7.4%	4.4/19	
	4	3.6%	2.5/7		-7.9%	5.2/21	
	5	3.3%	3.2/8		-8.9%	6.3/26	
medium	2	-8.8%	3.1/12	10.7%	-14.8%	3.0/8	15.5%
	3	-3.8%	3.8/16		-12.6%	5.7/25	
	4	-0.6%	3.8/14		-16.4%	5.8/17	
	5	1.4%	4.3/11		-18.5%	7.6/22	
big	2	-14.8%	3.0/10	6.4%	-29.8%	4.2/12	10.1%
	3	-13.4%	7.1/26		-25.8%	8.8/23	
	4	-7.7%	7.4/19		-28.2%	13.1/60	
	5	-3.1%	6.5/19		-28.2%	12.2/31	
huge	2	-22.8%	14.4/47	5.4%	-26.3%	22.1/76	7.1%
	3	-16.2%	11.5/21		-29.6%	12.4/20	
	4	-17.2%	5.5/13		-27.4%	8.2/16	
	5	2.0%	4.3/12		-14.4%	4.0/10	

Achieved coloring numbers in comparison to Nadara et al. Figure 2 illustrates the distribution of results for all turbocharging algorithms in a scatter plot. Data points below y-value zero mean that the turbocharging algorithm improves a bound on the weak r-coloring number of an instance. Among our approaches, we see that while WCOL-MERGE(r) performs well for small radii, IC-WCOL-RL(r) performs well for larger radii. Together with the analysis from above we can conclude that Nadara et al.'s approaches yield lower weak coloring numbers on average but there is fraction of roughly one half instances where our approaches supersede Nadara et al.'s, in particular if the computed weak coloring numbers are small. Overall, we improved bounds for 172 of the 334 considered instances after $t = 300s$. The resulting new bounds are lower by 5% on average over all instances. Follow-up investigations also showed that the relative improvement of turbocharging negatively correlates with the average degree of the graph, suggesting that our turbocharging algorithms work better for particularly sparse graphs.

As mentioned, our approach has the advantage that investing more time yields gradually better results. Figure 3 shows the number of instances for which a turbocharging approach improve weak coloring numbers compared to Nadara et al. after a specific time. That is, for a time t, the y-value of a line corresponds to the number of instances I such that $k_{I,\mathcal{A},\mathcal{H},t} < \mathsf{bestNadara}(I)$. The values for the line *best* are determined by taking the number of instances I where any of the turbocharging approaches is better than $\mathsf{bestNadara}(I)$. Interestingly, IC-WCOL-RL(r) starts off with more improved instances, however, after 3600 seconds, WCOL-MERGE(r) achieved more instances with smaller coloring numbers than Nadara et al. After 3600 seconds, IC-WCOL(r) was able to improve 24 instances compared to Nadara et al., WCOL-MERGE(r) 144 instances, and IC-WCOL-RL(r) 115 instances. Overall, we could improve upper bounds for 181 of the 334 instances with $t = 3600s$. These are nine more than for $t = 300s$.

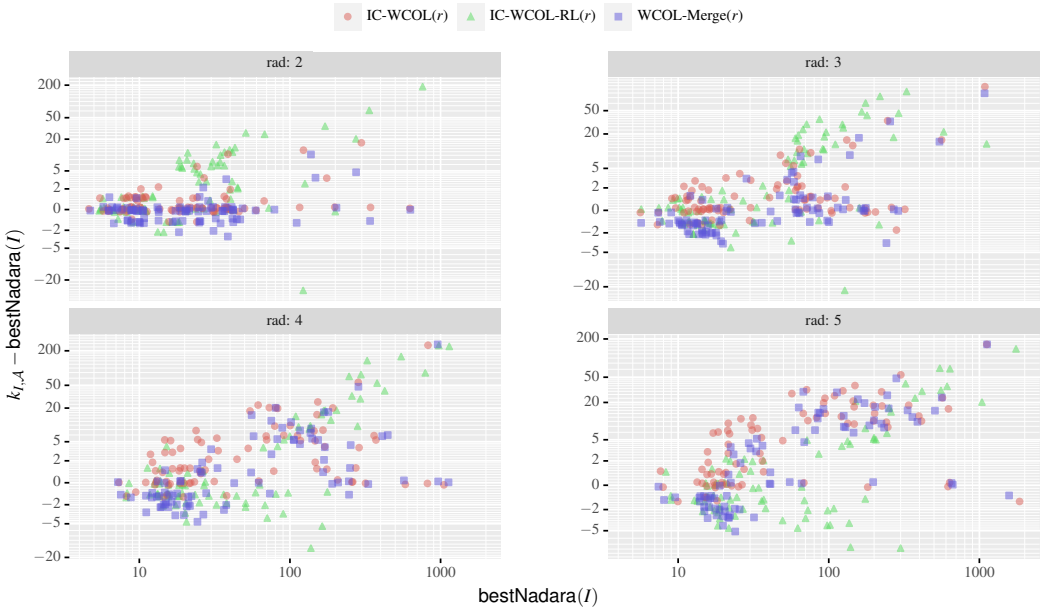

Figure 2 Scatter plot of the results for turbocharging algorithms broken down by radius. Each data point corresponds to an instance and a turbocharging algorithm. The x-value is the best weak coloring number achieved by Nadara et al. for this instance, the y-value is the difference of coloring numbers between the best weak coloring number achieved by Nadara et al. and the weak coloring number $k_{I,\mathcal{A}}$ achieved by the turbocharging algorithm $\mathcal{A}$ (minimum over both turbocharged heuristics). The x-axis is scaled logarithmically and the y-axis is scaled pseudo-logarithmically. Time limit: 300 s.

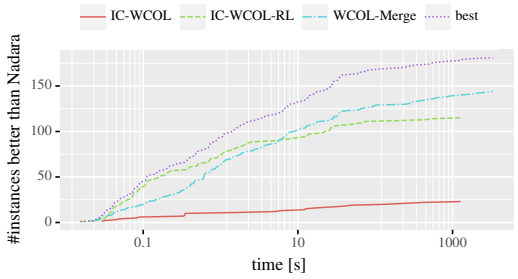

Figure 3 Line plot of the number of instances where a turbocharging approach achieves better coloring numbers than Nadara et al. over time. The time is scaled logarithmically.

5 Conclusion

On the theoretical side, we determined obstructions (running-time lower bounds) and promising avenues (a fixed-parameter algorithm) for applying the turbocharging framework to computing vertex orderings of small weak coloring numbers. On the experimental side, on a diverse set of instances each of the turbocharging approaches we use yields large improvements over the plain heuristics. This is most pronounced for the fixed-parameter turbocharging WCOL-Merge(r). Then we compared turbocharging to the best results gained by the seven heuristics that Nadara et al. [32] employed together with local search

procedures. Turbocharging so far yields on average larger weak coloring numbers than the best of the previous approaches. However, for 173 of the in total 334 instances, turbocharging outperforms all of the previous approaches combined. It works particularly well for small computed coloring numbers and sparse input instances.

References

1 Faisal N. Abu-Khzam, Shaowei Cai, Judith Egan, Peter Shaw, and Kai Wang. Turbo-charging dominating set with an FPT subroutine: Further improvements and experimental analysis. In T. V. Gopal, Gerhard Jäger, and Silvia Steila, editors, *Proceedings of the 14th Annual Conference on Theory and Applications of Models of Computation (TAMC 2017)*, volume 10185 of *Lecture Notes in Computer Science*, pages 59–70, 2017. doi:10.1007/978-3-319-55911-7_5.

2 Saeed Akhoondian Amiri, Patrice Ossona de Mendez, Roman Rabinovich, and Sebastian Siebertz. Distributed domination on graph classes of bounded expansion. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA 2018)*, pages 143–151. ACM, 2018. doi:10.1145/3210377.3210383.

3 Alex Pothen. The complexity of optimal elimination trees, 1988.

4 Hans L. Bodlaender, Jitender S. Deogun, Klaus. Jansen, Ton. Kloks, Dieter. Kratsch, Haiko. Müller, and Zsolt. Tuza. Rankings of graphs. *SIAM Journal on Discrete Mathematics*, 11(1):168–181, 1998. doi:10.1137/S0895480195282550.

5 Michael Breen-McKay, Brian Lavallee, and Blair D. Sullivan. Hardness of the generalized coloring numbers. *CoRR*, abs/2112.10562, 2021. arXiv:2112.10562.

6 Marco Bressan. Faster algorithms for counting subgraphs in sparse graphs. *Algorithmica*, 83(8):2578–2605, 2021. doi:10.1007/s00453-021-00811-0.

7 Marcin Briański, Piotr Micek, Michał Pilipczuk, and Michał T. Seweryn. Erdös–hajnal properties for powers of sparse graphs. *SIAM Journal on Discrete Mathematics*, 35(1):447–464, 2021. doi:10.1137/20M1342756.

8 Leizhen Cai, Siu Man Chan, and Siu On Chan. Random separation: A new method for solving fixed-cardinality optimization problems. In *Proceedings of the Second International Workshop on Parameterized and Exact Computation (IWPEC 2006)*, pages 239–250. Springer, 2006. doi:10.1007/11847250_22.

9 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

10 Erik D. Demaine, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, Somnath Sikdar, and Blair D. Sullivan. Structural sparsity of complex networks: Bounded expansion in random models and real-world graphs. *Journal of Computer and System Sciences*, 105:199–241, 2019. doi:10.1016/j.jcss.2019.05.004.

11 Alexander Dobler. Turbocharging heuristics for weak coloring numbers. Master's thesis, TU Wien, 2021. doi:10.34726/hss.2021.91580.

12 Alexander Dobler. Turbocharging heuristics for weak coloring numbers: Source code, November 2021. doi:10.5281/zenodo.5732923.

13 Alexander Dobler, Manuel Sorge, and Anaïs Villedieu. Turbocharging heuristics for weak coloring numbers. *CoRR*, abs/2203.03358, 2022. arXiv:2203.03358.

14 R. G. Downey, J. Egan, M. R. Fellows, F. A. Rosamond, and P. Shaw. Dynamic dominating set and turbo-charging greedy heuristics. *Tsinghua Science and Technology*, 19(4):329–337, 2014. doi:10.1109/TST.2014.6867515.

15 Jan Dreier. Lacon- and shrub-decompositions: A new characterization of first-order transductions of bounded expansion classes. In *Proceedings of the 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2021)*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470680.

16 Zdeněk Dvořák. Constant-factor approximation of the domination number in sparse graphs. *European Journal of Combinatorics*, 34(5):833–840, 2013. doi:10.1016/j.ejc.2012.12.004.

17 Kord Eickmeyer, Archontia C. Giannopoulou, Stephan Kreutzer, O.-joung Kwon, Michal Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Neighborhood complexity and kernelization for nowhere dense classes of graphs. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 63:1–63:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.63.

18 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *Journal of Experimental Algorithmics*, 18:3.1:3.1–3.1:3.21, 2013. doi:10.1145/2543629.

19 Serge Gaspers, Joachim Gudmundsson, Mitchell Jones, Julián Mestre, and Stefan Rümmele. Turbocharging treewidth heuristics. *Algorithmica*, 81(2):439–475, 2019. doi:10.1007/s00453-018-0499-1.

20 Martin Grohe, Stephan Kreutzer, Roman Rabinovich, Sebastian Siebertz, and Konstantinos S. Stavropoulos. Coloring and covering nowhere dense graphs. *SIAM Journal on Discrete Mathematics*, 32(4):2467–2481, 2018. doi:10.1137/18M1168753.

21 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *Journal of the ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

22 Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Adapting the bron–kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Social Network Analysis and Mining*, 7(1):35, 2017. doi:10.1007/s13278-017-0455-0.

23 Falk Hüffner, Christian Komusiewicz, and Manuel Sorge. Finding highly connected subgraphs. In Giuseppe F. Italiano, Tiziana Margaria-Steffen, Jaroslav Pokorný, Jean-Jacques Quisquater, and Roger Wattenhofer, editors, *SOFSEM 2015: Theory and Practice of Computer Science*, pages 254–265. Springer, 2015. doi:10.1007/978-3-662-46078-8_21.

24 Gwenaël Joret, Piotr Micek, Patrice Ossona de Mendez, and Veit Wiechert. Nowhere dense graph classes and dimension. *Combinatorica*, 39(5):1055–1079, 2019. doi:10.1007/s00493-019-3892-8.

25 Meir Katchalski, William McCuaig, and Suzanne Seager. Ordered colourings. *Discrete Mathematics*, 142(1):141–154, 1995. doi:10.1016/0012-365X(93)E0216-Q.

26 Hal A. Kierstead and Daqing Yang. Orderings on graphs and game coloring number. *Order*, 20(3):255–264, 2003. doi:10.1023/B:ORDE.0000026489.93166.cb.

27 Christian Komusiewicz and Manuel Sorge. An algorithmic framework for fixed-cardinality optimization in sparse graphs applied to dense subgraph problems. *Discrete Applied Mathematics*, 193:145–161, 2015. doi:10.1016/j.dam.2015.04.029.

28 Łukasz Kowalik, Marcin Mucha, Wojciech Nadara, Marcin Pilipczuk, Manuel Sorge, and Piotr Wygocki. The pace 2020 parameterized algorithms and computational experiments challenge: Treedepth. In Yixin Cao and Marcin Pilipczuk, editors, *Proceedings of the 15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*, volume 180 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:18. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.37.

29 O-joung Kwon, Michał Pilipczuk, and Sebastian Siebertz. On low rank-width colorings. *European Journal of Combinatorics*, 83:103002, 2020. doi:10.1016/j.ejc.2019.103002.

30 David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983. doi:10.1145/2402.322385.

31 R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002. doi:10.1126/science.298.5594.824.

32 Wojciech Nadara, Marcin Pilipczuk, Roman Rabinovich, Felix Reidl, and Sebastian Siebertz. Empirical evaluation of approximation algorithms for generalized graph coloring and uniform quasi-wideness. *ACM Journal of Experimental Algorithmics*, 24(1):2.6:1–2.6:34, 2019. doi:10.1145/3368630.

33 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Springer, 2012.

34 Jaroslav Nešetřil, Patrice Ossona de Mendez, Michał Pilipczuk, and Xuding Zhu. Clustering powers of sparse graphs. *The Electronic Journal of Combinatorics*, pages P4.17–P4.17, 2020. doi:10.37236/9417.

35 Michael P. O'Brien and Blair D. Sullivan. An experimental evaluation of a bounded expansion algorithmic pipeline. *arXiv:1712.06690 [cs]*, 2017. arXiv:1712.06690.

36 Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. Parameterized circuit complexity of model-checking on sparse structures. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 789–798. ACM, 2018. doi:10.1145/3209108.3209136.

37 Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web (WWW 2017)*, pages 1431–1440. International World Wide Web Conferences Steering Committee, 2017. doi:10.1145/3038912.3052597.

38 N. Przulj, D. G. Corneil, and I. Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515, 2004. doi:10.1093/bioinformatics/bth436.

39 Felix Reidl and Blair D. Sullivan. A color-avoiding approach to subgraph counting in bounded expansion classes. *arXiv:2001.05236 [cs]*, 2020. arXiv:2001.05236.

40 Felix Reidl, Fernando Sánchez Villaamil, and Konstantinos Stavropoulos. Characterising bounded expansion by neighbourhood complexity. *European Journal of Combinatorics*, 75:152–168, 2019. doi:10.1016/j.ejc.2018.08.001.

41 Sepp Hartung and Rolf Niedermeier. Incremental list coloring of graphs, parameterized by conservation. *Theoretical Computer Science*, 494:86–98, 2013. doi:10.1016/j.tcs.2012.12.049.

42 Sebastian Siebertz. Nowhere dense graph classes and algorithmic applications. a tutorial at highlights of logic, games and automata 2019. *arXiv:1909.06752 [cs]*, 2019. arXiv:1909.06752.